Presented at Fifth IASTED International Conference:
Expert Systems and Neural Networks
Honolulu, Hawaii, August 16-18, 1989

Determining Word Lexical Categories with a Multi-Layer Binary Perceptron

J.L. Cybulski[1], H.L. Ferrá[2], A. Kowalczyk[3] and J. Szymanski[4]

[1,3,4] *Artificial Intelligence Systems*
Telecom Research Laboratories
770 Blackburn Road
Clayton, Vic. 3168
Australia

[2] *Department of Mathematics*
Monash University
Clayton, Vic. 3168
Australia

Abstract. This paper discusses the method and an application of multi-layer perceptrons to the construction of a large syntactic lexicon of English words. Four different perceptron architectures are reviewed: real, integer, modulo and binary. A number of experiments in the construction of large perceptrons are discussed. Noise handling, weight rounding and weight factorisation are also considered.

Keywords: perceptron, lexicon, learning.

1. Introduction

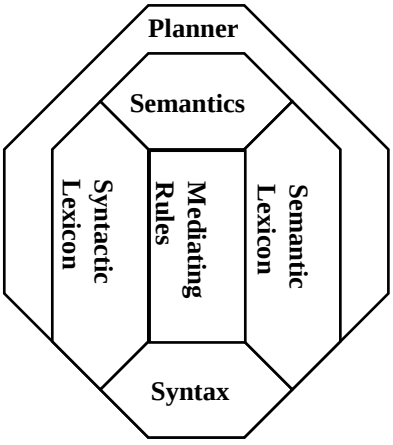


Figure 1 - COSIMO

COSIMO is a massively parallel natural language parser, being a part of a larger, plan-based, multi-media MEDICI system [8, 9]. COSIMO uses both syntax and semantics to fully analyse natural language queries into plan action schemata [13]. The system semantic grammar is based on the structure of plan schemata (Figure 1, see also [2, 3, 18]), whereas syntax is represented with a set of grammatical rules and a lexicon of word syntactic categories [7, 10]. Syntactic and semantic understanding in COSIMO is accomplished with a neural network based on the *interactive relaxation with decay* algorithm [7, 15, 19].

In a given knowledge domain the set of plan schemata, semantic features and the syntactic categories is usually small (around 200) and does not have a significant impact on the performance of the COSIMO neural network. The size of the syntactic lexicon, however, is well in excess of 10,000 words and the classification of input strings by the

interactive relaxation method causes serious computational problems. Taking into account the facts that the syntactic lexicon is static, the problem of word classification is deterministic, and the application has a relatively noiseless input, we consider a simple, multi-layer binary perceptron (MBP) as the first choice for efficient and compact implementation of a lexical analyzer [4]. Subsequently, we investigate real and modulo perceptrons for the noisy (misspelled) input especially.

The approach taken is an alternative to a more traditional hash table based lexicon. But what is more important for us, it is also a step to fully neural network parsing of natural language with the potential for very efficient hardware implementation and considerable noise immunity. Due to a high volume of exemplar data, the task of building such a lexicon is in itself a challenging neural network problem, and has significant implications for other non-linguistic classification problems.

Category	ID
infinitive verb	0
present verb	1
past verb	2
present participle verb	3
past participle verb	4
auxiliary verb	5
adjective	6
simple adverb	7
complex adverb	8
plural noun	9
singular noun	10
demonstrative pronoun	11
personal pronoun	12
quantifier pronoun	13
relative pronoun	14
wh-question pronoun	15
article	16
cardinal	17
ordinal	18
conjunction	19
preposition	20
proper nouns	21

2. Lexical Analysis of English Text in COSIMO

COSIMO's operation starts with lexical analysis of an input stream into word syntactic categories and semantic micro-features. The parser then interleaves the syntactic and semantic processing by spreading activation amongst its rules until the network settles into a steady state. At this point it is possible to select either the most likely syntactic or semantic interpretations of the input stream ordered by the level of their activation.

In general, a lexicon may be viewed as a mapping from a set of words into a set of lexical categories. In COSIMO we consider 22 categories, including nouns, verbs, adjectives and adverbs (see Fig. 2). Input to the lexicon-constructing program (cf. Fig. 3) consists of pairs (*word, classification*); the word is encoded in low-level word features (e.g. bits of ASCII code) and its classification is given by a string of 22 bits representing class membership indicators (1 means that a word is a class member, 0 otherwise; interpretation of syntactic categories is also given in a verbose form). Many words fall into multiple syntactic classes leading to ambiguous word interpretations, e.g. *abstract* may be classified as an adjective, a noun or an infinitive verb (cf. Fig 3). The resolution of these ambiguities is the very task of the syntactic and semantic parsers. In this context the implementation of word lexical

Word	Classification	Comment
abolish	10000000000000000000	+ infinitive_verb
abolished	00101000000000000000	+ past_verb+pastpart_verb
abolishes	01000000000000000000	+ present_verb
abolishing	00010000000000000000	+ prespart_verb
abolition	00000000001000000000	+ singular_noun
abolitions	00000000010000000000	+ plural_noun
abstract	10000010001000000000	+ adjective+infinitive_verb+single_noun
abstracted	00101000000000000000	+ past_verb+pastpart_verb
abstractly	00000001000000000000	+ adverb
abstracts	01000000010000000000	+ plural_noun+present_verb
any	00000000000001000000	+ quantifier
anyhow	00000000000000000010	+ preposition

Figure 3 - Sample Lexicon

classification, being the main theme of this paper, is an initial stage to practical syntactic analysis of natural language.

This paper focuses primarily on the syntactic lexical analysis of English texts. The methods and techniques for the initial semantic classification of words are similar and will be excluded from this discussion.

3. Multi-Layer Perceptrons

Architecture. In this paper, we consider a restricted class of perceptrons (Fig. 4). We assume all input units to have binary activation (i.e. 0 and 1 values), all hidden units to calculate logical conjunctions of subsets of inputs also resulting in their binary activations (note that conjunction can also be implemented with the "threshold logic"), and finally output units to calculate the weighted sum of all lower layer unit activations (note that the zero weight connections are routinely omitted).

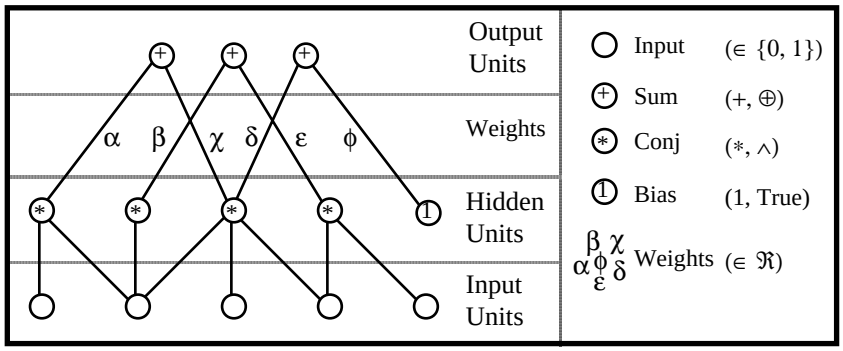


Fig. 4 - Perceptron architecture

Depending on the nature of these weights four different classes of perceptrons may be distinguished :-

(i) *Real perceptron.* Weights α, β, ... are real numbers, the conjunction in hidden units and the sum in the outputs are an arithmetic multiplication and a weighted sum of incoming activations, respectively. Thresholds on outputs may be added as necessary.

(ii) *Integer perceptron.* This is a real perceptron with integer weights.

(iii) *Modulo perceptron (m-perceptron).* In this case the real weights α, β,... belong to the open-closed interval (*-m, m*] , where *m* is a positive real number and the output sums are taken *mod 2*m* with representation in this interval.

(iv) *Binary perceptron (MBP).* This is a particular case of a modulo perceptron: the weights α, β,... are binary, 0 or 1, and the output sums are taken modulo 2, also resulting in the binary values (this corresponds to taking logical XOR).

Our perceptrons differ from the typical 2-layer perceptrons [14, 16], in particular, with respect to the following:

(i) the weights of links from input to hidden units are binary rather than real values,
(ii) our hidden units, performing logical conjunctions of their in-puts (or equivalently multiplication), can be viewed as a subclass of more general threshold units, and
(iii) the thresholds in output units are only optional.

It is not difficult to see that any deterministic transformation from a finite set of finite bit strings into a set of the finite strings of real numbers can be implemented by a real perceptron of the above form.

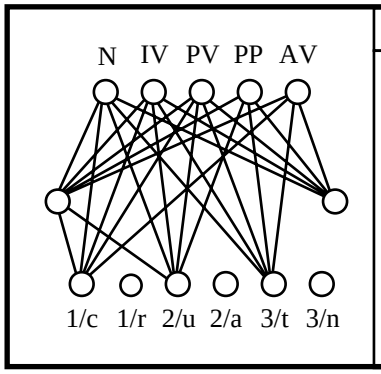
Moreover, if the difference between the largest and the smallest output of a real perceptron is not greater than *2*m+1*, then an *m*-perceptron can be used. Finally, if the output strings are binary, then a binary perceptron will serve that purpose. Thus, the structure is quite general. The thresholds in the outputs may be added as necessary. However, the explicit consideration of linear outputs in early stages of training allows more freedom and simplifies our training algorithms [4]. It may be shown that if such a "linear perceptron" is first trained in terms of minimizing its mean-square-error, then an imposition of a threshold on each of its outputs does not significantly affect the perceptron's overall performance [22].

	cat	cut	can	rat	rut	run	ran
1/c	1	1	1	0	0	0	0
1/r	0	0	0	1	1	1	1
2/u	0	1	0	0	1	1	0
2/a	1	0	1	1	0	0	1
3/t	1	1	0	1	1	0	0
3/n	0	0	1	0	0	1	1
noun	1	1	1	1	1	1	0
inf verb	0	1	1	0	1	1	0
past verb	0	1	0	0	0	0	1
past part	0	1	0	0	0	1	0
auxiliary	0	0	1	0	0	0	0

Fig 5 - Word classification decision table

Example. As an illustration, let us consider a simple decision table in Fig. 5 containing a set of seven words ("cat", "can", "cut", "rat", "ran", "rut" and "run") described by their letter positions ("1/c", "1/r", "2/a", "2/u", "3/t" and "3/n"). These words are classified into five syntactic categories ("noun", "infinitive", "past", "past participle" and "auxiliary verb"). In Figs. 6-8 we have examples of three different perceptrons implementing this decision table (note that hidden units are numbered from left to right and denoted with H_n).

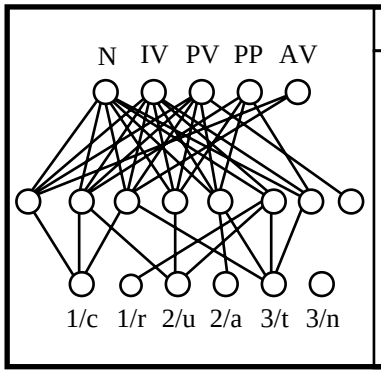
Note that the binary perceptron in Fig. 8, and also a 2-perceptron, and may be obtained by formally "factorising by 2" all weights of the the integer perceptron in Fig. 7 and real perceptron in Fig. 6, respectively. (The 2-perceptron is not shown since its architecture is identical to that of a real perceptron, with the exception of its weight H1-PV changed from 5/3 to -1/3.) In the conversion of an integer to binary perceptron the factorisation



Weights

1/c-N	1/2	3/t-PV	-1/3	H2-PV	2/3
1/c-IV	1/2	3/t-PP	-1/3	H2-PP	1/6
1/c-PV	-1/2	3/t-AV	-1/3	H2-AV	1/6
1/c-AV	1/2	H1-N	-2/3		
2/u-N	1/2	H1-IV	-1/3		
2/u-IV	1	H1-PV	5/3	*Threshold*	
2/u-PV	-1/2	H1-PP	2/3	*0.5*	
2/u-PP	1/2	H1-AV	-1/3		
3/t-N	1/3	H2-N	1/3		
3/t-IV	-1/3	H2-IV	1/6		

Fig. 6 - Real perceptron implementing word classification in Fig 5.



Weights

H1-N	1	H3-PV	1	H6-N	-4
H1-IV	1	H3-AV	-1	H6-IV	-4
H1-PV	-1	H4-N	1	H7-N	4
H1-AV	1	H4-IV	1	H7-IV	4
H2-N	-4	H4-PV	-1	H7-PP	-1
H2-IV	-4	H4-PP	1	H8-PV	1
H2-PV	1	H5-N	-3		
H2-PP	1	H5-IV	-4		
H3-N	-1	H5-PV	-1		
H3-IV	-1	H5-PP	1		

Figure 7 - Integer perceptron implementing word classification in Fig 5.

consists in retaining all the odd weighted links and discarding all of the even ones. This "factorisation approach" to generating modulo perceptrons is general and does not deteriorate the perceptron accuracy but, what is far more important, it decreases the magnitude and ratios of weights (cf. Section 5 and discussion of scaling problem in [16, pp 264-266]).

Training. Neural networks, including perceptrons, are usually generated from a set of selected exemplars by applying an appropriate training algorithm [1, 5, 6, 16].

Our algorithms aim at designing perceptron initial architecture by performing a heuristic controlled search for suitable hidden units, generally from their lowest to the higher order [4, 11, 12]. The crux of the approach is in a relatively quick algebra-based heuristic assessment of usefulness of the proposed new addition to the list of already selected units. Compared with fully distributed learning methods, our perceptron global optimisation reduces computational complexity and the perceptron generation time. Recalculation of synaptic weights need not be performed continuously; the weights are actually calculated once only, after the perceptron structure is determined. Fine tuning by some other retraining methods (like back-propagation [20]) or weight rounding may be used to further improve performance or to simplify the perceptron structure as desired [4].

As with any algorithm, our training method has its own drawbacks and limitations. The algorithm is memory intensive and is vulnerable to the number of exemplars and their encoding technique (cf. Section 4). It is worth to remark that the algorithm could be further enhanced by any preprocessing leading to the selection of a subset of essential inputs either by entropy or rough set methods [4, 11, 17].

In our opinion some other training algorithms, including back-propagation, could be considerably enhanced if started with initial "roughly correct" perceptron structure (e.g. generated by our methods), rather than a random one.

4. Experiments with Binary Perceptrons

In this section our approach to lexical analysis assumes noise-free input, a large dictionary (10,000 words), and the requirement of time/space efficiency. Our previous research [4] shows that MBP is especially useful in deterministic application domains, it may handle large sets of exemplars, and finally perceptron run-time

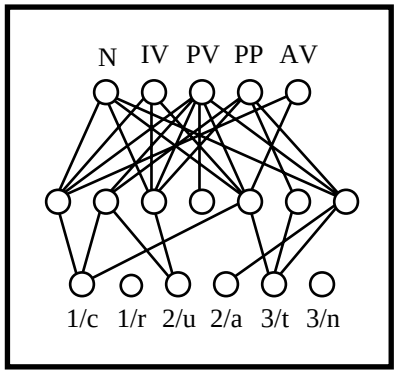


Figure 8 - MBP for classification in Fig. 5

efficiency may be guaranteed by their fully distributed and inherently parallel architecture. Hence, a multi-layer binary perceptron (MBP) seems a possible implementation platform for the construction and use of large lexical classifiers.

The performance of a perceptron-based lexicon is directly affected by a number of perceptron characteristics, e.g. the number and order of units in the perceptron input, hidden and output layers, amount of redundancy in learning patterns, the size of the training set and the total number of perceptron synapses. These factors, as experiments show, are in turn dependent on some of the application characteristics, e.g. pattern encoding method, clustering and ordering of word features, lexicon partitioning and the module size.

A number of experiments were set up to determine the best combination of perceptron characteristics.

Encoding methods. Lexicon representation, and thus the word and class encoding techniques, have an important role in the perceptron construction time and in its run-time performance. Firstly, input encoding determines the amount of redundancy on input and the order[1] of hidden units, both having tremendous impact on the perceptron generation time. Secondly, the word feature clustering and their ordering influences the size of a hidden layer and inter-connectedness between input and output layers.

In our experiments we aimed at producing perceptrons of a minimum number of hidden units of higher order in the shortest possible time. We looked at three different word encoding techniques :-

(i) *ASCII encoding*. The perceptron input consisted of 60 bits: the first 10 encoding the word length and the remaining 50 representing the significant ASCII bits of the word first 10 (or less) characters. The representation seemed to have very little redundancy and the intuition was that detection of word features would lead to the high order of hidden units.

(ii) *Map encoding*. The perceptron input consisted of 270 bits: the first 10 denoting the word length in characters, the remaining 260 divided into 26-bit letter maps representing actual characters of the word (1 on n-th position indicates the use

[1] The order of a unit is defined as the number of incoming synapses.

of n-th character in the alphabet). The sparse input seemed to guarantee the lower order of hidden units.

(iii) *Morphemic encoding*. Detection of word morphemic clusters and suffices facilitates easier word classification, e.g. suffices "-ed" or "-d" may indicate past or past participle verbs, "-ness" indicates a singular noun, "-al" an adjective, "-ing" present participle, etc. Reversing the order of characters at the perceptron input takes advantage of word characterisation by suffix by allowing the training algorithm to cluster the suffix features at early stages of processing. Note that the morphemic encoding could be applied to both ASCII and MAP codes.

The basic set of experiments were performed on a sample of 100-1000 randomly selected dictionary words, encoded in either ASCII or MAP code. The measured entities were the CPU time of perceptron generation on the Sun 3/60, the number of perceptron synapses, the number of hidden units, and the maximum order of output and hidden units (Fig. 9).

			Units							
			Hidden				Output			
Size	Synapses		Number		Max Order		Max Order		CPU Secs Sun 3/60	
	M	A	M	A	M	A	M	A	M	A
100	439	510	32	45	2	2	52	53	28	14
200	860	1227	89	145	2	2	97	102	75	53
300	1491	1922	151	241	2	2	160	154	163	126
400	1892	2821	225	345	2	3	202	213	313	431
500	3350	4397	318	442	2	3	264	266	545	765
600	4383	5367	403	542	2	3	307	317	890	1177
700	5587	6186	500	639	3	3	355	382	1928	1695
800	7106	7461	585	736	3	3	403	412	2677	2345
900	8548	9043	677	835	3	3	447	478	3629	3148
1000	9909	12201	735	898	3	3	506	505	4352	3782

Figure 9 - Comparison of MAP (M) and ASCII (A) encoding

Experiments show that ASCII encoding allows quicker perceptron generation but the resulting structures suffer from many deficiencies, e.g. high inter-layer connectivity, increased number of synapses, high order of hidden and output units. All this leads to increased size of run-time architectures and slower processing on sequential machines. Hence, the remaining experiments used sparse MAP encoding rather than an obvious ASCII coding. MORPHEMIC encoding imposed on top of the MAP resulted in 25-30%

improvement in the generation time and resulted in perceptrons of lower inter-layer connectivity.

It is also worth noting that performed experiments indicate the types of perceptron growth functions, i.e. their generation time is an exponential function of the size of its training set (conservatively $C^{\sqrt{size}}$, C • 1.3 for MAP). The number of synapses, hidden units, and the order of outputs grow linearly. The maximum order of hidden units is a function proportional to the input pattern redundancy (Fig. 10).

Test on Sun 3/60	*Module Size*	*Syn. Number*	*Hidden Units*	*Hidden Units*	*Output Order*	*CPU Secs*
Total	11747	86964	10177	N/A	N/A	46293
Max	1351	11604	1265	3	688	15584
Avg	470	3479	407	3	240	1852
a	811	6656	720	3	406	2848
b	546	4220	488	3	295	845
c	1199	9027	1125	3	607	7864
d	790	5226	708	3	396	2339
e	652	5163	552	3	310	1742
f	594	4730	535	3	302	1133
g	346	2061	278	3	173	405
h	365	2795	299	3	197	333
i	546	3332	478	3	275	1100
j	81	327	21	2	37	10
k	73	284	24	3	32	18
l	411	2666	343	3	215	449
m	477	3776	419	3	248	703
n	218	1953	160	2	120	59
o	269	2070	209	2	137	158
p	967	5959	895	3	514	3968
q	59	256	15	2	32	5
r	713	4128	618	3	343	4794
s	1351	11604	1265	3	688	15584
t	668	6532	602	3	350	1697
u	115	654	56	2	61	28
v	174	960	120	2	88	36
w	299	2498	244	3	154	172
x	0	0	0	0	0	0
y	21	84	3	2	11	1
z	2	3	0	0	2	0

Figure 10 - Lexicon modularisation

Size effects. Due to the exponential growth functions, it is impossible to construct a single perceptron performing lexical analysis for the 10,000 word dictionary. The two possible solutions to the problem are to either increase the number of perceptron layers (which necessitates changes to the training algorithms), or to split the dictionary into

modules of more manageable size. The second option seemed more viable and the dictionary was subdivided into 26 parts, each consisting of words starting with identical first letter.

As it may be deduced from Fig. 10, a perceptron-based lexicon may be realised in hardware by 2 layers of binary AND gates ($\log_2$MaxHiddenOrder), and 10 layers of binary XOR gates ($\log_2$MaxOutputOrder). The total number of AND and XOR gate layers may be reduced from 12 to 10 ($\log_2$AvgHiddenOrder + $\log_2$AvgOutputOrder) by performing a more uniform module partitioning, this means that word classification may be performed very efficiently within 10 machine cycles in parallel hardware.

Spell checking. One of the major problems with MBP-based lexicons is their inability to deal with misspellings (noise). Classification of misspelled words usually leads to random results. To prevent the problem from occurring we suggest a two-perceptron architecture (Fig. 11). The first of the two perceptrons decodes a word into its classification string (gray) and a word unique number (black). The second perceptron takes the number and encodes it back into the word. If the word appeared in the training set then both input and output should be identical. The two-perceptron detection of misspelled words has been proven to be feasible although expensive. Another possible solution to the problem is the use of noise-tolerant real perceptrons considered in the next section.

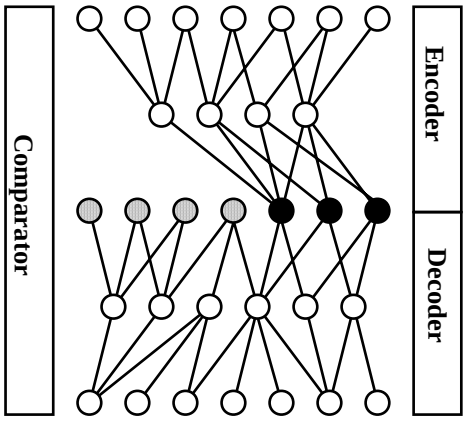


Figure 11 -
Misspelling detection

5. Experiments with Real Perceptrons

An alternative approach to lexical analysis, and handling misspellings in particular, is to use a real perceptron with thresholds (0.5) instead of a binary perceptron. Although such a perceptron is more difficult to implement (because of the real weights) it is more noise resilient (provided that the majority of its weights are small, i.e. « 1.0). As an initial attempt in this direction we generated two sets of real perceptrons :-

(i) for noise resistant classification in a dictionary of 100 words;

(ii) for classification of words in a dictionary of 1000 noise-free words.

Results are described below.

Noise-resistant encoding. A random noise was introduced to a dictionary of 100 words as follows. Each character of each word was assumed to be correct with confidence of 0.97, to be shifted by ±1 position in the alphabet with the probability of 0.02, and to be shifted by ±2 positions with the probability of 0.01. This corresponds to the probability of 0.74 of a 10 character word to be spelled correctly. A sample consisting of 20 repetitions of each of the 100 words (total of 2000 words and 474 different forms) was used in the basic experiment in the perceptron generation. Subsequently a sample of 10,000 words (previous 2000 plus 8000 unseen exemplars) was used for testing of generated perceptrons. Six of the trained real perceptrons are shown in rows 1-6 in Fig. 12.

For each of the perceptrons the table reports the total number of synapses, the number of terms of different order, distribution of weights, and the error rates for a training set (2,000 exemplars) and for a testing set (10,000 exemplars).

Note that although the error for the tests on the training set (2,000) generally decreases with addition of hidden units, the error for the test on 10,000 exemplars has a well pronounced minimum for perceptron no. 4. The weights of these perceptrons were

		Terms No		Weights				Error (%)		
No	Syn Ord.	Ord. 1	10^{-4} 2	0.18 –0.18	0.56 –0.56	1.8 –1.8	Max –5.6	for Ratio	for 2000	10000
1	611	50	0	399	193	19	0	9500.0	40.00	40.60
2	1780	149	0	758	664	254	4	19910.0	10.90	13.88
3	2763	182	66	1385	890	452	36	19951.7	1.90	5.33
4	3710	172	165	2049	1001	625	35	39040.0	0.10	3.87
5	4310	172	215	2412	1112	739	47	40931.0	0.15	4.95
6	4895	172	265	2344	1408	992	151	42350.0	0.00	6.37
7	2784	164	149	997	1101	649	37	39.0	0.10	4.01
8	2783	164	149	1008	1151	624	0	10.0	0.10	4.01

Figure 12 - Perceptrons for a dictionary of 100 misspelled words trained on 2,000 and tested on 10,000 exemplars

calculated with the precision of three decimal places. Further simplification of their structure can be achieved by :-

(i) *weight rounding* relaxing weight tolerance and reduces the scaling problem [16];

(ii) *mod 2 factorisation* of all weights and outputs, which reduces all their values into interval (-1, 1]; thresholds are implemented here by taking all factorised outputs with all values within the interval (-0.5, 0.5) to be 0, and within (-1, -0.5]≈[0.5, 1] to be 1.

Rounding the weights of perceptron no. 4 to one decimal place we obtained perceptron no. 7 (Fig 12). Successively the weights of this perceptron were factorised resulting in perceptron no. 8. This way we generated perceptrons with a much simpler structure, measured in terms of weight tolerance (an increase from 10^{-4} to 10^{-1}), maximal weights ratio (drop from 39040.0 down to 39.0 and 10.0, respectively), and the number of synapses (decrease from 3710 to 2784 and 2783, respectively).

		Terms			Weights				
No	Syn	Ord 1	Ord 2	Ord 3	0.001 0.18	0.18 0.56	0.56 1.8	Max Ratio	Error %
1	6701	158	141	0	6230	430	41	11499.7	21.6
2	9072	163	235	1	8250	735	87	12331.1	8.7
3	11451	171	324	4	10181	1116	154	17608.2	2.9
4	13002	171	424	4	11122	1654	226	16958.2	0.5
5	16275	172	523	4	13498	2367	350	17363.4	0.1

Figure 13 - Real and modulo perceptrons for encoding a dictionary of 1000 noise-free words

Encoding of noise-free dictionary with real perceptrons. A number of different real perceptrons were generated for a dictionary of 1000 randomly selected words, and five of them are described in Fig. 13. For each of the perceptrons the table reports the total number of synapses, the number of terms of different order, distribution of weights, and the error rates for the training set.

6. Conclusions

Experiments in building a lexical classifier using multi-layer perceptrons show the approach to be fully feasible.

When dealing with a relatively noise-free environment, an MPB classifier has certain advantages over real perceptrons, e.g. 100% accuracy, the simplicity of their architecture (binary weights) and potentially efficient hardware implementation (10 cycle response). However, real perceptrons are usually smaller, especially when 100% precision is sacrificed. A number of different encoding techniques were tested and the best results in the word representation was that of a sparse morphological input encoding (alphabet maps with reversed order). Due to the exponential time of the perceptron generation it was suggested to partition large training sets into roughly 1000 word modules, each used in construction of an independent binary perceptron.

The experiments show that real perceptrons are potentially better equipped to handle noise (misspellings). The obtained perceptrons showed a considerable ability for generalisation which manifested itself in their ability to correctly classify a number of misspelled, unseen cases (cf. test for 10000 in Fig. 12). The results show that in comparison to MBP, a large reduction in the number of hidden units is possible with only a marginal deterioration of performance (size 1000 in Fig. 9 vs Fig. 13). In particular, we observe that acceptance of the 2-3% loss in accuracy is compensated by considerable gain in reduced number of hidden units, and that in some cases removal of hidden units may even improve the performance (Fig. 12). We noticed that weight rounding and factorisation can further simplify perceptron structure, in particular by relaxing weight tolerance and reducing the ratio of the highest to lowest absolute weight value, without a significant decline in performance. It may also be worthwhile to mention that our past experience [4] shows that performance of real perceptrons could be further improved by employing additional retraining schemes (left for future research).

Overall, our approach to the perceptron training, consisting in generation of a roughly correct perceptron architecture followed by its fine tuning with rounding, factorisation and retraining methods, proved to be feasible and effective. The method was also shown to be of a particular value to the classification of very large sets of patterns.

7. Acknowledgement

We acknowledge the permission of Executive General Manager, Telecom Australia Research Laboratories to publish this paper.

8. References

[1] Ackley, D.H., Hinton, G.E. and Sejnowski, T.J. "A Learning Algorithm for Boltzmann Machines," Cognitive Science, Vol 9, 1985, pp 147-169.

[2] Bookman, L.A. "A Microfeature Based Scheme for Modelling Semantics," IJCAI 87, 1987, pp 611-614.

[3] Cottrell, G.W. and Small, S.L. "A Connectionist Scheme for Modelling Word Senses Disambiguation," Cognition and Brain Theory 6 (1), 1983, pp 89-120.

[4] Cybulski, J.L., Ferrá, H.L., Kowalczyk, A. and Szymanski, J. "Experiments with Multi-Layer Perceptrons," CSS Branch Paper 181, Telecom Australia, 1989.

[5] Grossberg, S. "How Does a Brain Build a Cognitive Code?," Psychological Review, Vol 87, 1980, pp 1-51.

[6] Hopfield, J.J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. of the National Academy of Sciences, Vol 79, 1982, pp 2554-2558.

[7] Howells, T. "VITAL: A Connectionist Parser," Cognitive Science, 1988, pp 18-25.

[8] Jennings, A. and Rowles, C. D "Capability Based Natural Language Understanding," Proc. Australian Joint AI Conf. 88, 1988, pp 419-430.

[9] Jennings, A. , Rowles, C. D and Kowalczyk, A. "Natural Language Understanding in the MEDICI Project," Proc. Int. Conf. on Comp. and Inf., Toronto, Canada, 1989.

[10] Jones, M.A. "Feedback as a Coindexing Mechanism in Connectioninst Architecture," IJCAI 87, 1987, pp 602-610.

[11] Kowalczyk, A. "Empirical Induction Algorithm for Approximate Reasoning," CSS Branch Paper 178, Telecom Australia, 1989.

[12] Kowalczyk, A. "Optimisation of Decision Operator," CSS Branch Paper 179, Telecom Australia, 1989.

[13] Leckie, C. and Zukerman, I. "Planning in Cooperative Domains: A Case Study," CSS Branch Paper 178, Telecom Australia, 1989.

[14] Lippmann, R.P. "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, IEEE, April, 1987, pp 4-22.

[15] McClelland, J.L. and Rumelhart, D.E. "An Interactive Activation Model of Context Effects in Letter Perception," Psychological Reviews 88, 1981, pp 375-407.

[16] Minsky, M.L. and Pappert, S.A., Perceptrons, Cambridge, Massachusetts: The MIT Press, 1969. (Expanded edition, 1988.)

[17] Pawlak, Z., Wong, S.K.M. and Ziarko, W. "Rough Sets: Probablistic Versus Deterministic Approach," Technical Report, Department of Computer Science, University of Regina, 1987.

[18] Pollack, J.B. and Waltz, D.L. "Parallel Interpretation of Natural Language," Proc. Int. Conf. on Fifth Generation Computer Systems, ICOT, 1984.

[19] Reggia, J.A. "Properties of Competition-Based Activation Mechanism in Neuromimetic Network Models," Proc. IEEE First Int. Conf. on Neural Networks, 1987.

[20] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. "Learning Internal Representations by Error Propagation," in Rumelhart and McClelland, 1986, pp 318-362.

[21] Rumelhart, D.E. and McClelland, J.L. (eds) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume1: Foundations, Cambridge, Massachusetts: The MIT Press, 1986.

[22] Widrow, B. and Hoff, M.E. "Adaptive Switching Circuits," 1960 IRE WESCON Convention Record, New York: IRE, 1960, pp 96-104.